CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Journal Paper

A Real-Time Software Defined Networking Framework for Next-Generation Industrial Networks

Luis Moutinho ; Paulo Pedreiras ; Luis Almeida

CISTER-TR-200110

A Real-Time Software Defined Networking Framework for Next-Generation Industrial Networks

Luis Moutinho ; Paulo Pedreiras ; Luis Almeida

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

Industry 4.0 brings in a whole set of new requirements to engineering industrial systems, with notorious impact at the networking layer. A key challenge posed by Industry 4.0 is the operational flexibility needed to support on-the-fly reconfiguration of production cells, stations, and machines. At the networking layer, this flexibility implies dynamic packet handling, scheduling, and dispatching. SoftwareDefined Networking (SDN) provides this level of flexibility in the general Local Area Network (LAN) domain. However, its application in the industry has been hindered by a lack of support for real-time services. This paper addresses this limitation, proposing an extended SDN OpenFlow framework that includes realtime services, leveraging existing real-time data plane Ethernet technologies. We show the OpenFlow enhancements, a real-time SDN controller, and experimental validation and performance assessment. Using a proof-of-concept prototype with 3 switches and cycles of 250;cs, we could achieve 1;cs jitter on timetriggered traffic and a reconfiguration time between operational modes below 10ms.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 8, 2019, accepted November 3, 2019, date of publication November 7, 2019, date of current version November 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2952242

A Real-Time Software Defined Networking Framework for Next-Generation Industrial Networks

LUIS MOUTINHO[1,2], (Member, IEEE), PAULO PEDREIRAS[1,3], (Senior Member, IEEE), AND LUIS ALMEIDA[4,5], (Senior Member, IEEE)

[1]Instituto de Telecomunicações (IT), Aveiro 3810-193, Portugal
[2]Escola Superior de Tecnologia e Gestão de Águeda (ESTGA), 3750-127 Águeda, Portugal
[3]Electronics, Telecommunications and Informatics Department, University of Aveiro, 3810-193 Aveiro, Portugal
[4]Electrical and Computer Engineering Department, Faculty of Engineering, University of Porto (FEUP), 4200-465 Porto, Portugal
[5]Research Centre on Real-Time and Embedded Computing Systems (CISTER), 4249-015 Porto, Portugal

Corresponding author: Luis Moutinho (lems@ua.pt)

This work was supported in part by the Portuguese Government through a grant of the Operational Programme of Competitivity and Internationalization of Portugal 2020 under Grant PRODUTECH II SIF, POCI-01-0247-FEDER-024541, and in part by the Research Centre Instituto de Telecomunicações under Grant UID/EEA/50008/2013.

ABSTRACT Industry 4.0 brings in a whole set of new requirements to engineering industrial systems, with notorious impact at the networking layer. A key challenge posed by Industry 4.0 is the operational flexibility needed to support on-the-fly reconfiguration of production cells, stations, and machines. At the networking layer, this flexibility implies dynamic packet handling, scheduling, and dispatching. Software-Defined Networking (SDN) provides this level of flexibility in the general Local Area Network (LAN) domain. However, its application in the industry has been hindered by a lack of support for real-time services. This paper addresses this limitation, proposing an extended SDN OpenFlow framework that includes real-time services, leveraging existing real-time data plane Ethernet technologies. We show the OpenFlow enhancements, a real-time SDN controller, and experimental validation and performance assessment. Using a proof-of-concept prototype with 3 switches and cycles of $250\mu s$, we could achieve $1\mu s$ jitter on time-triggered traffic and a reconfiguration time between operational modes below $10ms$.

INDEX TERMS HaRTES, Industry 4.0, OpenFlow, real-time communications, software-defined networking.

I. INTRODUCTION

Industry 4.0 refers to the current revolution towards seamlessly integrating physical objects, humans, and smart production systems in a sophisticated information network known as the Industrial Internet of Things (IIoT) [1]. Examples include factories that autonomously adapt to supply-demand fluctuations or product variations, that use self-organizing logistics and/or self-diagnosing machines [2], [3]. This revolution will bring efficiency gains and cost reductions, leading to improvements in product quality, manufacturing and logistics planning, and customer satisfaction [4].

Industry 4.0 is based on data availability and interoperability throughout the entire value chain, from devices to logistics and, ultimately, the consumer. This integration poses unprecedented networking requirements on flexibility, heterogeneity, reconfigurability, and timeliness [1], [3].

The first three requirements can be met with current network management frameworks such as Software-Defined Networking (SDN) [5] that decouples network control and data planes, enabling a logically centralized management of network resources, with fine granularity and high flexibility [5], [6]. However, SDN was developed for general purpose LANs. Thus, its traffic model favors throughput with best-effort policies, possibly imposing bandwidth constraints or establishing fixed priorities, while forsaking communications timeliness. This is incompatible with many industrial applications that require strict predictability, timeliness, and fault tolerance. In turn, industrial networking technologies were specifically designed to meet strict timeliness

The associate editor coordinating the review of this manuscript and approving it for publication was Yulei Wu.

requirements, particularly very low latency and jitter, while showing little concern for throughput and online reconfiguration.

In this paper, we contribute to close such a gap by extending SDN with adequate real-time mechanisms. The related scientific literature [6]–[8] reports the use of SDN on industrial networks but at the conceptual level, only, under simulated or emulated scenarios, with limitations in scalability, predictability and/or flexibility [7]. The work in [9] appears to be the first step to add explicit support for flexible real-time communication within SDN by leveraging the flexibility of HaRTES real-time Ethernet switches. We build on this work to present a full-featured SDN framework that includes the real-time extensions and respective Application Programming Interface (API), together with a corresponding real-time SDN controller.

To the best of our knowledge, our work is the first cohesive real-time SDN framework that includes all needed architectural components to address the flexibility, timeliness, and heterogeneity challenges of Industry 4.0. Our contributions are the following: i) SDN OpenFlow real-time data plane extensions (a superset of those in [9]); and ii) a corresponding real-time SDN controller. We also present an experimental validation and performance evaluation of the whole framework using a proof-of-concept prototype inspired by Industry 4.0.

The paper structure is the following. Section II presents network requirements imposed by Industry 4.0. Section III discusses previous work on SDN within industrial networks. Section IV presents our framework architecture and Section V introduces the proposed OpenFlow real-time extensions. Section VI discusses the mapping of the OpenFlow extensions on the real-time data plane while Section VII presents the real-time SDN controller. Section VIII shows the framework experimental validation and Section IX concludes the paper.

II. INDUSTRY 4.0 NETWORK REQUIREMENTS

Industry 4.0 will bring a host of new applications and paradigms based on full interoperability between all stages of the value chain and all levels of the production system, from Enterprise Resource Planning (ERP) systems to Manufacturing Execution Systems (MES) and the control of machines on the shop floor [1], [3]. At the same time, the traditional ISA-95 [10] automation pyramid will gradually converge to a network of decentralized machines and services with a high degree of autonomy and cooperation [11]. These changes impose new requirements on industrial networks [3] concerning, for example, security, timeliness, and operational flexibility. In this work, we focus on the requirements concerning heterogeneity, timeliness, and management, namely:

- **R1: Support simultaneous applications with heterogeneous QoS requirements.** Table 1 presents typical Quality of Service (QoS) requirements, particularly timing, of classes of applications that will co-exist in Industry 4.0 and which need to be fulfilled jointly [3], [12].

TABLE 1. Typical QoS (timing-related) requirements of Industry 4.0 applications [3], [12].

	Motion Control	Cell Control	Augmented Reality	HMI*
Latency / Cycle Time	$250\mu s$ to $1ms$	$1ms$	$10ms$	$100ms$
Jitter	$\leqslant 1\mu s$	$\leqslant 1ms$	- - -	- - -
Data rate	kbit/s	k-Mbit/s	M-Gbit/s	M-Gbit/s

- - - Not specified
*Human-Machine Interface

- **R2: Meeting precisely applications QoS requirements.** Over-provisioning of network resources must be avoided to achieve efficient use of the network capacity.
- **R3: Dynamic message and reservation sets.** Order-controlled production and adaptable factories [2], leveraged by Multi-Agent Systems (MAS) and Service-Oriented Architectures (SOA) [13], [14] imply changing the communication requirements online. This change is itself time-constrained, *e.g.*, in the seconds range [13].
- **R4: Real-time network monitoring and diagnostics.** Precise communications and network state monitoring is key to react promptly to changes in communication requirements and potentially harmful situations [3], [7].
- **R5: Consistent set of open management tools.** The current practice of using vendor-specific and non-interoperable sets of management tools limits heterogeneity and jeopardizes the benefits of Industry 4.0.

Requirements R1-R2 can be fulfilled to some extent by deploying multiple network technologies, but this approach jeopardizes the interoperability across the value-chain and the realization of requirements R3-R5. Conversely, SDN frameworks essentially fulfill requirements R4-R5 but exhibit limitations on the realization of R1-R3. These limitations, and current efforts to overcome them are discussed in the next section.

III. RELATED WORK

The scientific literature reports a modest number of contributions on the use of SDN in industrial scenarios, most of which evaluate and assess the suitability of SDN for such context and identify its weaknesses and limitations. Notwithstanding, a few proposals address the development of formal methods to analyze its performance and ways to extend its functionality. In this section, we review the most relevant scientific contributions in this area.

A. QUALITATIVE EVALUATIONS

The work in [7] identifies open challenges behind the use of SDN in future industrial networks. Many of those challenges stem from limitations in expressing industry-grade requirements, *e.g.* QoS and real-time timing guarantees (R1-R2), using the existing southbound API. Ehrlich *et al.* [8] present a similar study, identifying a set of ten requirements of future industrial network management frameworks (a super-

set of R1-R5) and comparing against the current capabilities of SDN. They conclude that SDN fulfills a subset of the requirements, namely independence from underlying network technologies, support for online (re)configurations, and security-related features (essentially R3-R5), but still fails in several aspects such as the inability to set QoS provisioning and monitoring (R1-R2) due to unsuitable southbound APIs. Kálmán [6] overviews the most relevant characteristics of SDN and identifies possible ways to apply it in industrial Ethernet scenarios. The work refers significant advantages in network deployment, monitoring, and dynamic management brought by SDN's centralized control (misses R1, R2).

B. PERFORMANCE ANALYSIS

Thiele and Ernst [15] performed a formal analysis of the OpenFlow protocol (OFP) envisioning its deployment on a Time-Sensitive Networking (TSN) Ethernet network. Their analysis considers the communication with the controller, the network topology and the scalability limits for real-time usage. Their work focuses on the actual management flow, not on the real-time traffic performance. Nonetheless, the formal analysis shows that attaining latency values below 50ms is possible in such systems (limited fulfillment of R1). In [16], Herlich *et al.* highlight the possible gains in supporting arbitrary network topology, dynamic (re)configuration, and fast fail-over, that can be obtained by applying SDN to real-time Ethernet networks. They resort to experiments using a virtual platform with OpenFlow deployed on Ethernet POWERLINK (EPL) networks showing fail-over without packet losses. However, EPL is intrinsically bandwidth inefficient (collides with R2).

C. FUNCTIONALITY EXTENSION AND USE IN REAL-TIME NETWORKS

In [17], Ternon *et al.* investigate how the Flexible Time-Triggered (FTT) paradigm can be instantiated on standard OpenFlow hardware making it suitable for real-time scenarios. The paper presents a new protocol, FTT-OpenFlow, that exploits the OpenFlow capabilities to enhance the response time of sporadic real-time traffic and shows analytically that the proposed solution meets the requirements of an avionics scenario. FTT-OpenFlow is also shown to improve the performance of non-real-time traffic. However, this work does not address its applicability to multi-hop scenarios (limited fulfillment of R1-R2). In [18], Nayak *et al.* exploit the logical centralization of SDN to build a global view of the network and use it to compute the routes and a transmission schedule to constrain in-network queuing of time-triggered traffic. The central controller uses OpenFlow to configure the data plane devices and enforce the computed routes and transmits the schedule to the source nodes that synchronize their transmissions according to the assigned temporal slots. Despite obtaining low latency and jitter for time-triggered traffic, the coexistence of sporadic real-time traffic is not addressed (collides with R1). Ahmed *et al.* [19] propose SDPROFINET, an SDN architecture for smart factories using PROFINET networks. The approach, similar to that of [18], consists in having the central controller collect information of the network and configuring the data plane devices according to the routes of the PROFINET data channels. This architecture improves network management, but its flexibility is constrained by PROFINET, *e.g.*, static offline scheduling and limited support for event-triggered traffic. (collides with R1-R3). In [20], Ishimori *et al.* propose a hierarchical scheduling approach, similar to that of Linux Traffic Control (TC), to overcome the limitations of the First-In First-Out (FIFO) policies employed by OpenFlow devices. It supports HTB (Hierarchical Token Bucket), RED (Randomly Early Detection), and SFQ (Stochastic Fair Queuing). These scheduling policies provide bandwidth-based traffic shaping, only. Thus, despite improving performance, explicit support to real-time traffic is still poor (collides with R1, R2). Finally, Silva *et al.* [9] propose an extension to SDN to add explicit support for flexible real-time communication using HaRTES switches in the data plane. It allows dynamic management of both time-/event-triggered real-time traffic and non-real-time traffic, but without continued QoS guarantees due to the absence of a central admission control with schedulability analysis (limited support to R1, R3).

The above discussion shows that existing approaches to using SDN in the industry are either conceptual, only, lacking proper validation, and/or do not address all Industry 4.0 network requirements stated in Sec. II. Conversely, we build on the work in [9] to propose a complete SDN framework that meets all those requirements and which is validated with a prototype implementation in a relevant use-case.

IV. SYSTEM ARCHITECTURE

The proposed framework follows the standard SDN paradigm in which the control plane is enacted by a logically centralized controller that is responsible for the management of the data plane, comprising one or more interconnected network devices. FIGURE 1 displays an example of the system architecture for the proposed framework, with the data plane instantiated on a network of real-time switches.

The controller keeps information regarding the state and topology of the whole network, and interacts with

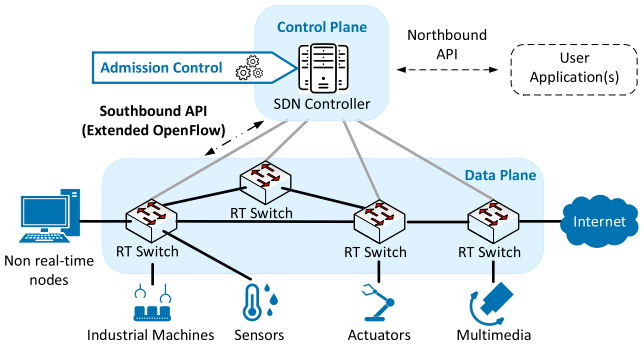

FIGURE 1. System architecture of the proposed real-time SDN framework with an example data plane using a network of real-time switches.

applications through the so-called northbound interface which can be built, for example, on top of RESTful APIs. Applications may ask for the reservation of network resources for both non- and real-time traffic flows by issuing requests that specify the desired QoS parameters and network endpoints. The development of this northbound API is not addressed in this work, being assumed that all necessary information regarding applications is available within the controller. Upon receiving requests, the controller runs an admission control algorithm to verify if the system has enough network resources and that real-time guarantees, for both new and existing flows, are met. If a flow is admitted, the controller sets the entire data path with adequate configurations, otherwise, the request is denied.

The control plane interacts with data plane devices using the OpenFlow switch protocol (OFP) [21], the *de facto* standard southbound API, configuring the necessary services and installing rules that dictate, for example, how traffic is to be forwarded throughout the network. Data plane devices process traffic according to the rules instantiated by the controller. To fulfil the Industry 4.0 networking requirements in Sec. II, the data plane network has to support: i) concurrent time-triggered, event-triggered and standard non-real-time traffic (requirement R1); ii) traffic policing and confinement mechanisms (R1); iii) hierarchical real-time reservations for event-triggered traffic (R1, R2); iv) efficient use of network resources (R2); and v) online flow management without service disruption (R3). Partial fulfillment of these data plane requirements will imply limited support of the Industry 4.0 networking requirements.

Finally, for the sake of simplicity, performance, and security, we consider that each data plane device connects to the controller using a dedicated port. Alternatively, a specific reservation in an ordinary port can be used for configuring the data plane switching devices, as also specified by OFP.

V. THE REAL-TIME OPENFLOW EXTENSIONS

OpenFlow switches receive frames at ingress ports and send them to a set of *Flow Tables* (FIGURE 2) for processing [21]. Each flow table contains a set of *Flow Entries* with: i) a priority to sort matching; ii) filters to identify incoming frames; iii) associated instructions; and iv) fields for statistics. Filters can address several frame fields, *e.g.*, Ethernet and IPv4 addresses. When a frame matches the filters of a given flow entry, the associated instructions are performed. These may change the list of actions associated with the frame, the *Action Set*, or explicitly direct it to a subsequent flow table for additional processing. Once the frame reaches the last table or is not directed to a subsequent one by the matched entry, its current action set is executed. Consequently, the frame can be sent to a group table (see below), forwarded to a given OpenFlow egress port, or dropped. The protocol defines egress ports as: i) *physical*, *i.e.*, switch hardware ports; ii) *reserved*, *i.e.* logical ports, for specific predefined processing, such as forwarding packets to a set of physical ports; and iii) *logical*, for processing methods that are defined outside the protocol. A total of $2^{24} - 1$ ports are defined to be freely assigned as either physical or logical.

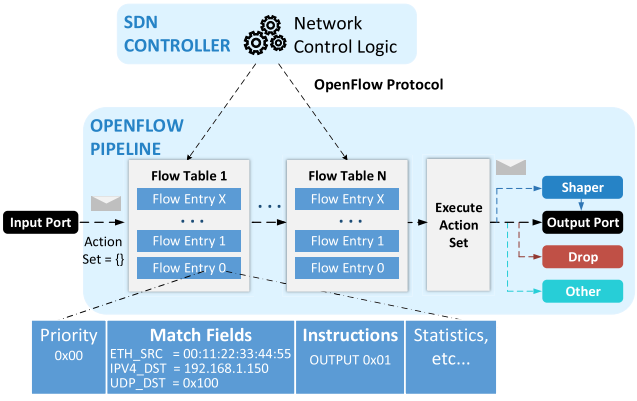

FIGURE 2. Overview of an OpenFlow pipeline.

Group tables contain a subset of instructions similar to those of flow tables, with similar outcomes. Optionally, devices can provide a *Meter Table* with different traffic shapers, designated *Meters*, that can be configured and targeted by actions to constrain the traffic rate before forwarding it to egress ports.

SDN controllers may use a set of messages specified by the OpenFlow protocol to configure basic device capabilities, *e.g.* the number of flow tables, filters and instructions of entries belonging to flow and group tables, meters, and access multiple utility functions, *e.g.* retrieve state information.

These services enable a powerful and dynamic management of traffic forwarding throughout the network. However, OpenFlow relies solely on meters to provide QoS, restricting traffic according to bandwidth thresholds expressed as maximum frames/s or kbit/s. Hence, the offered guarantees for event-triggered (ET) traffic are severely constrained and there is no support for time-triggered (TT) traffic. Moreover, the existing OpenFlow API is devoid of suitable messages and parameters to express and configure real-time services. To address these issues, we specify a Real-Time OpenFlow (RTOF) add-on (FIGURE 3) that makes enabled data plane devices support two logical domains: i) the standard OpenFlow domain; and ii) the real-time domain, with its own set of real-time services and API. This dual-stack approach allows seamless integration of real-time services and facilitates porting the real-time extensions to newer versions of OFP.

The RTOF add-on assigns a subset of logical ports to internal queues, each one statically linked to a unique real-time flow. SDN controllers may use the real-time API to query devices and discover the Unique Identifier (UID), the associated logical port and the traffic type (time- or event-triggered) of all supported flows, configure the flows real-time parameters, *e.g.* period and frame size, and access utility functions to, for example, retrieve state information. The RTOF add-on adopts the typical attributes used to specify real-time services, not being tied to a specific underlying communication protocol. Consequently, data-plane devices have to translate and enforce the necessary reservations, *e.g.* configure scheduling

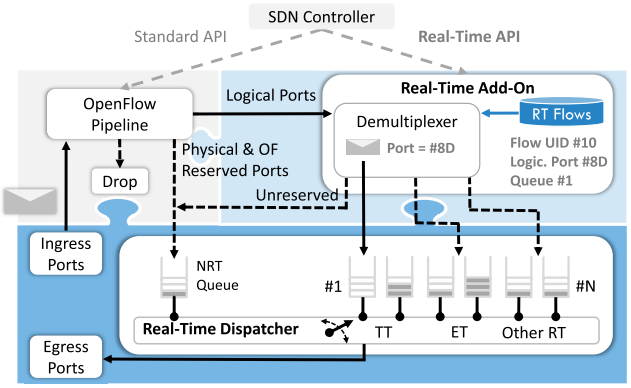

FIGURE 3. Overview of the devised Real-Time OpenFlow (RTOF) add-on.

TABLE 2. Real-time OpenFlow (RTOF) add-on API.

Message	Description	Sender
RT_ST_ADD	Add flow to the add-on database.	C
*ST_MOD	Modify properties of existing flow.	C
*ST_DEL	Remove flow from add-on database.	C
*ST_LIST_REQ	Get UID of existing flow (by type).	C
*ST_LIST_REP	Reply to RT_ST_LIST_REQ.	S
*ST_PROP_REQ	Get properties of a given flow.	C
*ST_PROP_REP	Reply to RT_ST_PROP_REQ.	S
*ST_REMOVED	Notification of flow removal events.	S
*ST_STATS_REQ	Get properties/statistics of all flows.	C
*ST_STATS_REP	Reply to RT_ST_STATS_REQ.	S
*ST_CAP_REQ	List supported real-time services.	C
*ST_CAP_REP	Reply to RT_ST_CAP_REQ.	S

Senders: Controller (C), Switch (S). * stands for "RT_".

tables, according to the configured flows' properties and deny requests that cannot be satisfied. Moreover, data plane devices must also implement a dispatcher that prioritizes real-time traffic over non-real-time (NRT).

To add a real-time flow in the framework, a controller must access each switch across the flow's route and i) *configure the appropriate real-time services* by using the RTOF API to register the flow and its properties, and ii) *configure the OpenFlow pipeline* by using the standard API to set flow entries with adequate matching rules and instructions to redirect the real-time flows to the corresponding logical ports.

The implemented RTOF API, briefly listed in Table 2, supersedes that of [9] and allows SDN controllers to remove and modify real-time flows, list all registered flows and their attributes, and get monitoring information. As in [9], the API is built on top of OpenFlow *Experimenter* messages, which are standard messages that vendors can use to offer additional functionality. For illustration purposes, the structure of the RT_ST_ADD operation message is depicted in FIGURE 4. It comprises an header common to all OpenFlow messages (*ofp_header*), fields to identify the vendor (*experimenter*) and the operation (*experience*), as well as a payload (*experimenter_data*). The *experience* field indicates the API operation while the payload carries the respective parameters. The other messages have a similar structure.

VI. MEDIATING RTOF AND THE REAL-TIME DATA PLANE

The proper operation of all RTOF extensions relies on the capability of data plane devices to meet all data plane

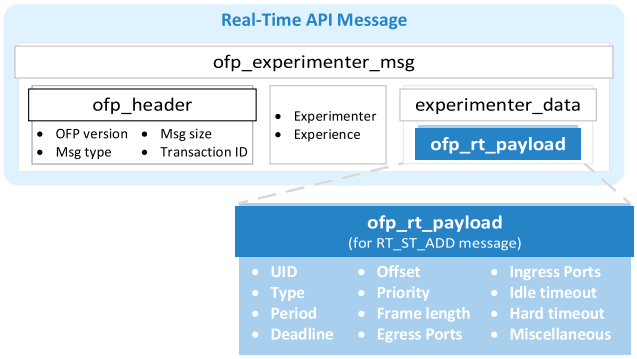

FIGURE 4. Structure of a RTOF API message for a RT_ST_ADD operation.

requirements referred at the end of Sec. IV. Data plane devices that do not support all those requirements can be used, but the real-time services that depend on the absent features shall be degraded or even unsupported. For this reason, in this paper, we use HaRTES switches [22] that implement a flexible real-time data plane that meets all the requirements thus supporting all proposed RTOF extensions.

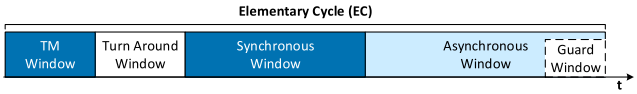

FIGURE 5. Structure of an Elementary Cycle (EC) in HaRTES.

The HaRTES switches are currently implemented on NetFPGAs 1G [23] and follow the master/multi-slave FTT paradigm, integrating an embedded FTT master that coordinates communications in fixed-duration time intervals called Elementary Cycles (ECs) (FIGURE 5). Each EC starts with the local broadcast (confined to each switch) of the trigger message (TM) that synchronizes nodes and conveys the schedule of time-triggered (TT) traffic for that EC. Next, a turn-around window allows nodes to interpret the TM and prepare the scheduled TT transmissions within the synchronous window. Event-triggered (ET) traffic, managed by a hierarchical server-based scheduling mechanism, is confined to the asynchronous window. Non-real-time transmissions are handled as background traffic within the asynchronous window, too. A guard window at the end of the EC prevents new asynchronous transmissions to be triggered, effectively avoiding EC overruns.

When multiple HaRTES switches are used, their ECs are synchronized to enforce a global time-triggered framework. This is done using a Global Trigger Message sent by an elected switch, or by using clock synchronization or even an out-of-band signal communicated through digital ports. The current prototype resorts to the latter approach. Multi-hop time-triggered forwarding is accomplished by scheduling that traffic simultaneously in all traversed switches, a method known as Reduced Buffering Scheme (RBS) [24].

On the other hand, HaRTES does not follow the SDN paradigm, thus lacking the basic services required by OpenFlow. Hence, integrating HaRTES into the real-time SDN

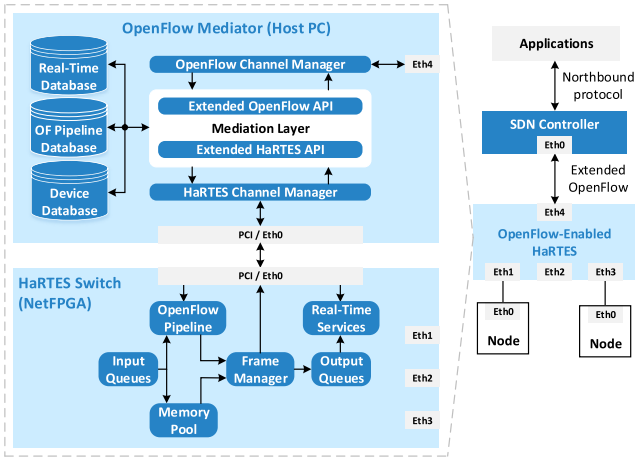

FIGURE 6. Architecture of the OpenFlow-enabled HaRTES.

framework requires: i) removing its native network control functions, *e.g.*, learning forwarding tables; ii) implementing the OpenFlow pipeline; and iii) creating an OpenFlow-compatible interface to communicate with controllers and to translate requests from the OpenFlow domain to proper HaRTES commands.

In the extended HaRTES platform (FIGURE 6), the OpenFlow pipeline and other processing entities, *e.g.* group tables, are implemented on the HaRTES hardware platform. The translation between the OpenFlow and HaRTES domains is implemented in software by the *OpenFlow Mediator* (OFM), a daemon that executes in a support (host) computer. Services for the communication with OpenFlow controllers, as well as databases to store current system configurations, are also provided by the OFM. The communication between the computer and HaRTES can be accomplished through a PCI interface or, alternatively, by using a dedicated Ethernet port (Eth0 in FIGURE 6).

In this architecture, frames sent by regular nodes are received by HaRTES's Ethernet ports and stored in the central memory pool. At the same time, a pointer to each frame is also sent to the OpenFlow pipeline. The actions resulting from the pipeline processing are then handled by a frame manager that populates the output queues accordingly. HaRTES's real-time services dispatch traffic stored in output queues according to the configured reservations. Non-real-time traffic, *e.g.*, standard OpenFlow, is also managed by the aforementioned services, in a best-effort way. The OpenFlow pipeline and HaRTES's real-time services are configured by the OFM upon receiving requests from SDN controllers.

A. OPENFLOW PIPELINE

The OpenFlow pipeline (FIGURE 7) is instantiated at each physical port to allow parallel execution without resource contention.

Frames received in each port are buffered byte-by-byte by the MAC controller into the respective FIFO input queue. A Finite-State Machine (FSM), the *Header Extractor* (HE),

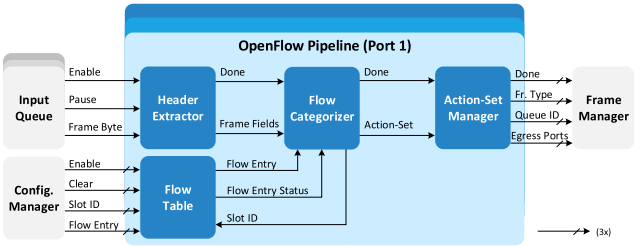

FIGURE 7. HaRTES's OpenFlow pipeline.

retrieves buffered bytes in a cut-through fashion, *i.e.*, while the frame is still being received, and extracts the fields that are essential for flow matching. The FSM operation is paced by the *Enable* and *Pause* signals that indicate when there are valid stored bytes (Pause = '0') of an incoming frame (Enable = '1'). Once all fields are extracted, the *Header Extractor* sends a request (Done = '1') to the *Flow Categorizer* (FC) that starts matching them against the filters of configured flow entries (Flow Entry Status = '1'). Flow entries are stored at the *Flow Table* in individual slots. Each slot is identified by a unique *Slot ID* and associated with a priority level (higher IDs mean higher priority levels). The FC checks configured entries one by one in descending priority until there is a match or the table-miss flow entry, *i.e.* the last table entry with 0 priority (Slot ID = 0), is reached. The action-set of matched entries is sent to the *Action-Set Manager* (A-SM) that decodes it into queuing instructions for the *Frame Manager* (FM), *e.g.*, frame type, egress queue ID and target output ports. Finally, the *Frame Manager* stores frames in their output queues. The *Configuration Manager* simultaneously configures and replicates flow entries on all pipelines according to commands received through the HaRTES API.

Concerning performance, the current OpenFlow pipeline processes a received frame within a time interval P_{RT} that is mainly dominated by (i) the pipeline clock speed f_{clk}, (ii) the maximum number of supported flow entries N_{FE} and (iii) the time to receive all the supported flow matching fields t_{HE}. The bound for P_{RT} can be computed following Eq. 1, where the rightmost term accounts for the number of clock cycles (4) related to output logic between modules (HE, FC, and A-SM) and the decoding of operations by the A-SM module.

$$P_{RT} = t_{HE} + N_{FE} * \frac{1}{f_{clk}} + 4 * \frac{1}{f_{clk}} \quad (1)$$

Presently, the pipeline in our prototype supports a single flow table with 32 flow entries and is capable of filtering frames based on Ethernet addresses, EtherType, IPv4 addresses, IP Protocol, and UDP/TCP ports. Moreover, only drop and output OpenFlow actions are currently supported for the action-set. Output actions support all physical, reserved and logical port types. Additional tables and flow entries can be deployed at the expense of more FPGA resources and a slight modification of the *Flow Categorizer* FSM. Each pipeline consumes approximately 3% of the total RAM blocks and logical slices provided by the NetFPGA's

Virtex-II Pro 50 FPGA, being inexpensively implemented in more recent FPGAs. Moreover, traffic is not subject to additional delay by the OpenFlow processing since the worst-case value for P_{RT} ($f_{clk} = 62.5MHz$) is significantly lower than the time to receive a minimum-sized frame. To configure the OpenFlow pipelines, we developed extensions to the HaRTES API that allow the OpenFlow Mediator to enable, modify, or disable any given flow entry.

B. OPENFLOW MEDIATOR

The OFM (FIGURE 6) deals with the complex management aspects of the OpenFlow protocol that are hard to support in hardware, implementing the logical services of the RTOF add-on. Specifically, the mediator i) establishes and maintains all dedicated OpenFlow channels for communication with controllers, ii) interprets OpenFlow requests and respective messages, iii) translates OpenFlow requests to suitable data plane calls (HaRTES API) for proper configuration of the OpenFlow pipeline and real-time services and iv) keeps databases with system state and capabilities information. There are three databases, i) the Real-Time Database containing the properties of all installed real-time flows, ii) the OpenFlow Pipeline Database, containing a synchronized image of the installed configurations on the devices pipelines and iii) the Device Database with information of the underlying device capabilities, *e.g.*, number of flow entries and a list of supported real-time logical ports.

The current OFM prototype is based on the open-source code of the popular CPqD OpenFlow 1.3 Software Switch (ofsoftswitch13) [25], stripped of switching capabilities and enhanced with the real-time extensions proposed in Sec. V.

C. DATA PLANE LAYER CONSIDERATIONS

The choice for HaRTES switches as data plane devices was obvious since, to the best of the authors' knowledge, these are the only Ethernet switches that currently support all the referred data plane requirements (end of Sec. IV). This good match allows supporting all RTOF extensions to OpenFlow proposed in this paper and thus, fulfill all the Industry 4.0 network requirements (Sec. II). Moreover, the open implementation of HaRTES in FPGA technology allows a smooth integration of OpenFlow pipelines, leading to high performance.

Nevertheless, the RTOF extensions can be used with virtually all real-time data plane technologies, but with corresponding limitations. Each technology will require a specific mediator to translate the RTOF API to its own configuration interface, *e.g.*, NETCONF. A specific OpenFlow pipeline(s) implementation will be needed, too. For example, the preliminary work in [26] shows the RTOF extensions supported on a WiFi data plane with real-time capabilities based on the Wireless MultiMedia (WMM) profile and using LinuxTC in the end-nodes for per-flow dynamic network reservations. All RTOF services are supported except for TT traffic. However, the OpenFlow pipeline implementation in software, within a soft Access Point, incurs a significant performance penalty.

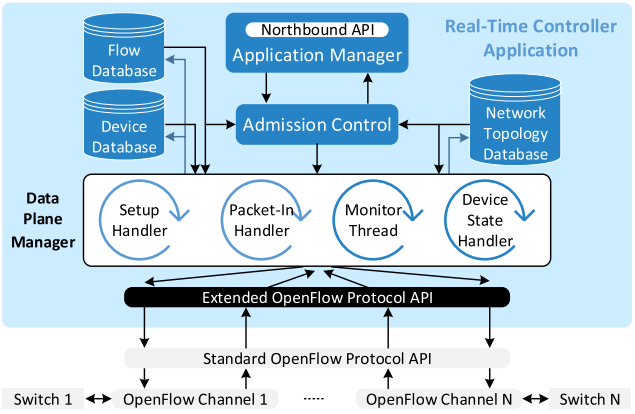

FIGURE 8. Architecture of the SDN real-time controller.

When implementing RTOF over common real-time Ethernet technologies, the limitations will depend on the specific protocol. For example, PROFINET IRT and TTEthernet will not support the dynamic reconfiguration of individual flows nor hierarchical reservations for ET traffic, while AFDX and EtherNet/IP will not support TT services. TSN will present more subtle limitations. Dynamic scheduling of its TT traffic is possible but cumbersome due to the repeating timetables technique used therein and the ET traffic is handled with a small set of shaped priority classes and not per-flow. A discussion on TSN *vs* SDN can be found in [27].

Finally, the performance of an RTOF mapping to a data plane technology, *e.g.* in latency and jitter, depends largely on the OpenFlow pipeline implementation. Meeting the requirements in Table 1 will most likely need a hardware implementation with tight integration with the device's architecture.

VII. REAL-TIME SDN CONTROLLER

Although there are several commercial and open-source SDN controllers [5], *e.g.* Floodlight [28], Faucet [29], OpenDaylight [30] and ONOS [31], these mainly address applications related to network virtualization and load-balancing services, not providing the services required to manage networks with strict timeliness requirements, *e.g.*, admission control with real-time schedulability analysis. Therefore, we designed and integrated in the proposed real-time SDN framework a controller that supports the Real-Time OpenFlow extensions described before (FIGURE 8).

A. CONTROLLER DESIGN AND DEPLOYMENT

We implemented the real-time SDN controller using Ryu [32], a popular open-source framework written in Python and tailored to the development of network management and control applications. Ryu was chosen due to i) its support for the most recent OpenFlow versions, ii) its component-based architecture that allows building applications on top of bare-bone services with great flexibility while significantly reducing the complexity and computational overheads that otherwise would result from the inclusion of unnecessary services, and iii) its support for Python-written modules, enabling quick prototyping with reasonable computing performance.

The real-time controller application can communicate with one or more data plane switching devices using the RTOF API. This API and OpenFlow channels are all provided and managed by Ryu bare-bone services. Devices first connecting to the controller are handled by the *Setup Handler* at the *Data Plane Manager*. This handler creates an entry for the device in the *Device Database* and performs basic configurations to the device's OpenFlow services, *e.g.* installs a flow entry to send traffic without filtering rules to the controller (an operation known as *packet-in*). The *Device Database* maintains, for each device, information on i) basic capabilities, *e.g.* number of Ethernet ports, OpenFlow tables and supported instructions, ii) installed OpenFlow configurations, *e.g.* flow entries, iii) statistic information retrieved from OpenFlow services, and iv) state information, *e.g.* the state of Ethernet ports. The *Monitor Thread* periodically polls statistics and installed OpenFlow configurations from each device, while the *Device State Handler* receives events sent by devices regarding state changes. Combined, they keep the *Device Database* up-to-date and consistent. The *Packet-In Handler* is responsible for handling packet-in transactions and act accordingly, *e.g.* consult user applications or install flow entries to drop traffic from that flow. The *Network Topology Database* stores information regarding network topology and is maintained by the *Setup* and the *Device State Handler*. We use NetworkX [33] to model the network as a directed graph, with switches interfaces as vertices and logical links as edges, and perform the necessary operations, *e.g.* discover a route for a given flow. Finally, the *Flow Database* (Γ) stores information regarding the properties of the N_s real-time flows (S_i) in the network (Eq. 2), namely the transmission period or minimum inter-arrival time (T_i), the transmission time of its maximum sized frame (C_i), the priority (P_i), the initial offset (O_i) ignored for event-triggered traffic, and relative deadline or maximum latency (D_i). The set of links $\mathcal{L}_i$ that flow S_i crosses through is also included.

$$\Gamma = \{S_i(T_i, C_i, P_i, O_i, D_i, \mathcal{L}_i), i = 1, 2, .., N_s\} \quad (2)$$

The *Application Manager* receives requests from user applications, *e.g.* through the northbound API, and forwards them to the *Admission Control*. This unit consults the *Flow Database* and *Network Topology Database* to retrieve traffic and topology information to run admission tests and determine if a given flow can be accepted in the network. Upon positive outcome, the request is forwarded to the *Data Plane Manager* that installs the adequate configurations on the data plane using the RTOF API. If the flow is rejected, a notification is sent to the *Application Manager* and relayed to the respective application. The *Data Plane Manager* continuously updates all databases according to changes in the network and state of connected devices.

B. ADMISSION CONTROL

The *Admission Control* (AC) runs a schedulability test whenever an application requests a change in its flows to make sure the available resources allow meeting the requirements, *e.g.* timing, of the new configuration without jeopardizing the corresponding requirements of the running configuration. For this purpose, the AC computes an upper bound of the end-to-end worst-case flow response time (R), *i.e.* the time lapse between the instant in which a respective frame becomes ready at the sender's interface and the instant in which its reception at the receiver's interface terminates. Currently, the response times are computed as described in [24] for HaRTES multi-hop networks in RBS mode but other analytical techniques, *e.g.* Network Calculus, can be used, too.

Algorithm 1 presents the AC operation upon receiving a request for a new flow (or an aggregated set of new flows).

Algorithm 1 Admission Control for a New Flow S_{req}

```
1:  if isRequestValid(S_req) ≠ true then
2:      return REJECT
3:  end if
4:
5:  Links[ ] ← getNetworkPath(S_req.src, S_req.dst)
6:  if Links = ∅ then
7:      return REJECT
8:  end if
9:
10: if calcFlowR(S_req, Links, FlowsDB) > S_req.Deadline
    then
11:     return REJECT
12: end if
13:
14: FlowsDB.insert(S_req)
15: for each S ∈ FlowsDB , S ≠ S_req do
16:     if calcFlowR(S, S.Links, FlowsDB) > S.Deadline
    then
17:         FlowsDB.remove(S_req)
18:         return REJECT
19:     end if
20: end for
21:
22: return ACCEPT
23:
24: function calcFlowR(S_i)
25:     R_i ← 0
26:     a ← b ← 1
27:     while b ≤ Nł_i do
28:         R_{i,a,b} = ⌈ responseTimeCalc(S_i, L_{i,a,b}) / ΔEC ⌉
29:         if a ≠ b ∧ R_{i,a,b} ≠ R_{i,a,(b−1)} then
30:             R_i ← R_i + R_{i,a,(b−1)}
31:             a ← b
32:         else
33:             b ← b + 1
34:         end if
35:     end while
36:     return (R_i + R_{i,a,(b−1)})
37: end function
```

First, the AC unit starts by checking the request correctness, *e.g.* number and consistency of attributes, and then

computes the corresponding flow path (lines 1-8). If the flow path or attributes are deemed invalid, the request is rejected, otherwise, R is estimated for the new flow(s) (lines 10-12). If R overcomes the maximum latency (deadline) of the new flow(s), the AC unit rejects the request (line 11). Else, the new flow(s) is inserted in the Flows Database (line 14). Note that R is also computed for every other flow in the system (lines 15-20) to assess potential interference caused by the new flow(s). Again, if R overcomes any deadline, the AC stops the analysis, removes the new flow(s), and rejects the request (lines 16-18), otherwise, the new flow(s) is accepted (line 22).

R for flow S_i is computed following the function *calcFlowR* (lines 24-37). In short, the function computes the response time of traffic frames from the source to the sink node while checking whether frames are buffered and sent in the following EC by any switch along the route [24]. The computation is performed on a link-by-link basis, from the first link in the L_i set (line 26) until its last link ($N\text{ł}_i$) (line 27). Buffering is detected when the response time (line 28) for the ingress and the egress link of the same switch, *i.e.* the next-to-last and the last element in $L_{i,a,b}$ differ (line 29). In this case, the computed response time up to the ingress link is added to the total response time and the computation starts a new step from the egress link of the buffering switch (lines 30 and 31). If the frame is not buffered, the next link in L_i is added to the response time calculation of the next loop iteration (line 33). The total response time is obtained when the main loop reaches the last link in the route of S_i (line 36).

The response time of S_i for a given link subset $L_{i,a,b}$ (line 28) is based on the classical response time analysis, iterating Eq. 3.

$$R_{i,a,b}(x) = \frac{C_i}{\alpha_{i,a,b}} + I_{i,a,b}(x) + B_{i,a,b}(x) + SD_{i,a,b}(x) \qquad (3)$$

The calculation considers the transmission time C_i of the S_i frame itself, as well as three types of interference from other flows sharing common links, namely (i) $I_{i,a,b}$ the interference from messages with higher or equal priority, (ii) $B_{i,a,b}$ the blocking from messages with lower priority, and (iii) $SD_{i,a,b}$ the total switching delay for the flow across the entire route. As frame transmissions are confined to the synchronous and asynchronous windows, an inflation factor $\alpha_{i,a,b}$ is used to account for possible inserted idle-time within the respective window. The response time is obtained when the iterative process converges, *i.e.* $R_{i,a,b}(x) = R_{i,a,b}(x-1)$, starting with $R_{i,a,b}(0) = \frac{C_i}{\alpha_{i,a,b}}$. The analysis is explained in detail in [24].

VIII. EXPERIMENTAL VALIDATION

This section establishes the feasibility of the proposed real-time SDN framework through an actual implementation in a realistic use-case comprising heterogeneous traffic flows. This implementation also provides insight into the level of performance that can be expected with this framework, particularly against the timing requirements in Table 1. The real-time SDN controller runs on an i7-4770 CPU with 8GB DDR3 RAM and Arch Linux 4.20.7 OS and the dataplane

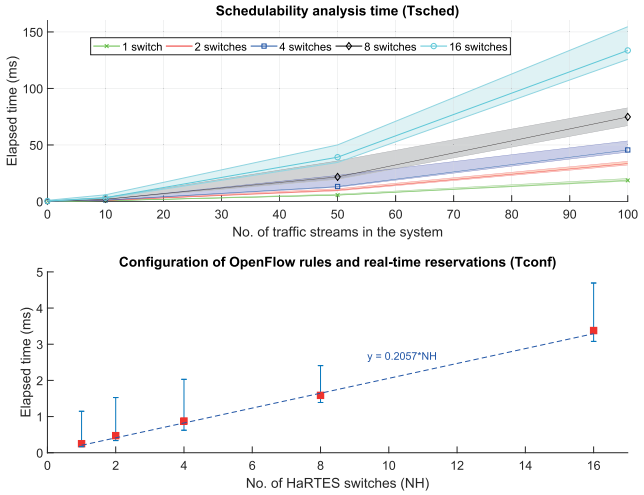

FIGURE 9. Timings for admission control (T_{sched}) and data plane configuration (T_{conf}) procedures (1000 samples per point with the shades/intervals representing the min-max range observed).

uses several HaRTES switches enhanced with OpenFlow pipelines. The experiments and their results are described throughout this section.

A. RECONFIGURATION RESPONSIVENESS

In this experiment, we connect our Ryu-based SDN controller to multiple switches ($N_H = 1, 2, 4, 8, 16$), and measure the time it takes to perform the admission control analysis and configure all devices upon a request to admit a new flow in the system. All switches are interconnected following a line topology and the new flow traverses the entire network, from an ingress port in the first switch to an egress port in the last one. This is a worst-case scenario in which all devices must be properly configured. A set of random flows ($N_S = 0$, 10, 50, or 100 flows) is previously installed in the system to assess the load effect on the schedulability analysis algorithm. As the analysis stops computing as soon as a deadline of a given flow is violated, to capture the worst-case execution time of the admission control, we consider long enough deadlines for all flows. This ensures that the algorithm always computes the response time for the new and installed flows without breaking the process in the middle.

FIGURE 9 shows the observed execution times. The schedulability test time (T_{sched}) grows significantly with the number of flows since the algorithm complexity is $O(N_S * (N_H + 1))$ [24]. Note that $N_H + 1$ is the number of traversed links. Conversely, the network configuration time (T_{conf}) is independent of the number of flows (N_S) and directly proportional to the devices to be configured (N_H). Eq. 4 shows an empirical linear model. Thus, even for a reasonably sized network (16 switches and 100 flows), our framework can respond to reconfiguration requests in less than 170ms, which is significantly less than the desired reconfiguration time for small industrial production systems, *i.e.* seconds range, referred in [13].

$$T_{conf} = 0.2057 * N_H \ \ (ms) \qquad (4)$$

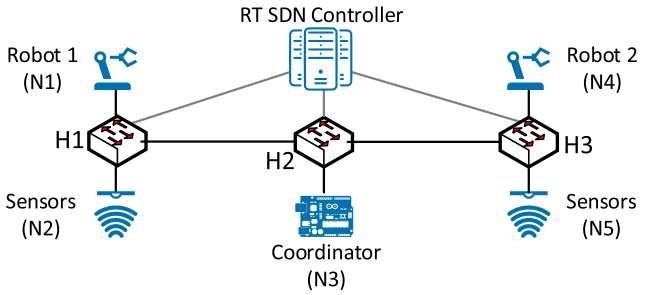

FIGURE 10. Network setup for the test application.

B. TRAFFIC TIMELINESS

To assess the traffic timeliness, we set up the flows of a smart distributed robotic cell in a representative Industry 4.0 application scenario, adapted from [34]. We considered the number of ports that our Ethernet switches have (4) and we also considered significantly smaller periods of the flows to increase the network load. The setup, described in FIGURE 10, comprises the real-time SDN controller connected to three HaRTES switches via dedicated links plus five application nodes. The SDN controller also executes the three OpenFlow mediators (OFM1, OFM2, and OFM3) for the three HaRTES switches (H1, H2, H3). The links are 100 Mbit/s and the Elementary Cycle is 250μs long. The length of the *Synchronous* and *Asynchronous* windows is set to 80 and 160μs, respectively. The *TM* and *Turn Around Window* take 10μs. The ECs of the three switches are synchronized via an explicit signal applied to specific digital ports.

The coordinator (N3) controls robots 1 and 2 (N1 and N4, resp.) by transmitting a set of periodic commands according to data received from each robot and from a cluster of sensors (nodes N2 and N5). Table 3 shows the parameters of the flows used in the experiment, where T is the period or minimum inter-arrival time, PL the payload, and O the initial offset for TT traffic. Flows 1 to 8 carry control data (TT traffic) while flows 9 to 18 convey monitoring data and alarm events (ET traffic). Finally, flows 19 to 24 are non-real-time traffic, namely statistics and production logs. The RT flows are scheduled with Rate-Monotonic priorities but always before the NRT ones. The application also exhibits two distinct operational modes according to the actual number of robots in use. Thus, *mode A* uses robot 1, only, while *mode B* uses robots 1 and 2 concurrently. The mode changes occur online, without service interruption.

All nodes are implemented in FPGAs for high precision traffic generation. A hardware sniffer, namely a Hilscher netANALYZER NANL-C500-RE, captures packets in multiple links and timestamps them with nanosecond resolution. In particular, it allows measuring the time between the first bit of a frame in the sender link and the corresponding first bit in the receiving link, to which we add the frame transmission time to obtain the response time. Thus, these times do not account for possible delays in the nodes' network interfaces. Table 4 shows the observed message response times and jitter between consecutive frames of the same flow. The worst-case response time (R) estimated by the SDN controller, in ECs, is shown in the table, too.

TABLE 3. Setup communication requirements (adapted from [34]).

Flow ID	Source	Sink	Type	T (μs)	PL (Bytes)	O (ECs)
1-3	N3	N1	TT	500	80	0
4-6	N3	N4	TT	500	80	1
7	N2	N3	TT	250	160	0
8	N5	N3	TT	250	160	0
9,10	N1	N3	ET	1000	390	—
11,12	N4	N3	ET	1000	390	—
13	N2	N3	ET	1000	390	—
14	N5	N3	ET	1000	390	—
15,16	N2	N1	ET	250	46	—
17,18	N5	N4	ET	250	46	—
19	N1	N3	NRT	1000	750	—
20	N4	N3	NRT	1000	750	—
21	N2	N1	NRT	1000	750	—
22	N5	N4	NRT	1000	750	—
23	N3	N1	NRT	500	750	—
24	N3	N4	NRT	500	750	—

TABLE 4. Real-time traffic performance.

Flow ID	Response time (μs) Min.	Mean	Max.	Jitter (μs)	R^1
Mode A					
1-3	31.1-31.2	31.2-31.3	31.4-32.5	0.9-1.0	1
7	50.3	50.5	50.6	0.9	1
9-10	105.6-106.4	197.5-219.9	305.1-338.9	269.5-338.0	3
13	105.7	166.1	409.4	476.9	3
15-16	14.4-14.5	49.8-52.9	149.8-149.9	179.1-179.9	1
Mode B					
1-6	32.1-32.2	32.2-32.3	32.4-32.5	0.8-1.0	2
7	50.3	50.5	50.7	0.9	1
8	65.0	65.9	66.6	1.0	1
9-10	105.7-106.4	202.6-232.9	340.6-390.6	258.2-322.7	4
11	105.7	258.2	475.1	496.0	4
12	104.6	341.7	461.1	393.4	4
13-14	103.6-105.6	209.9-211.0	621.0-654.3	690.3-821.6	4
15-18	14.3-14.5	51.1-54.2	189.3-199.3	306.0-329.1	1

500 000 samples per flow. EC is 250μs long.
[1] Estimated worst-case response time (No. of ECs).

The results show that, in both modes, control (TT) traffic is forwarded with latency values below 70μs (within 1 EC) and jitter at or below 1μs. These values fulfill the most stringent requirements (motion control applications) in Table 1. This is a consequence of the tight synchronization among nodes and proper EC configuration. An interesting detail can be observed with flow 8 in mode B, which suffers interference from flow 7 in the last link, increasing its response time. It is also important to realize that HaRTES follows a synchronous approach with the resolution of ECs, not controlling explicitly the order of transmission within the Synchronous Window. Thus, to achieve very low jitter in the TT traffic, care must be taken in defining which messages are scheduled together in the same EC, adjusting their periods or offsets if needed. This is common in the construction of TT schedules.

For ET traffic, we observe that all frames are delivered within their minimum inter-arrival time (implicit deadlines). We also note that the maximum response times are within the bounds produced by the analysis with relatively low pessimism. In mode A the analysis is tight for 2 of the 5 ET flows with a 1 EC excess in the remaining 3, while in mode B the analysis is tight for 4 of the 10 ET flows, with 1 EC

excess in other 2 and 2 ECs excess in the remaining 4. The observed jitter is relatively high for this type of traffic due to the possibility that frames are randomly blocked by ongoing transmissions in switches egress links.

Regarding mode reconfiguration, the mean and maximum time to change from mode A to mode B, *i.e.* adding robot 2 reservations, is 7.78*ms* and 9.37*ms*, respectively, while from mode B back to mode A, *i.e.* deleting robot 2 reservations, is 2.98*ms* and 3.73*ms*. These values were obtained with 1000 mode change cycles.

C. ON FULFILLING INDUSTRY 4.0 NETWORK REQUIREMENTS

The experimental results reported before indicate that the conceptualized SDN framework can fulfill all the Industry 4.0 network requirements identified in Sec. II. With the HaRTES-based data plane, we support the coexistence of multiple applications with distinct QoS requirements while ensuring full guaranteed timeliness (requirement R1) with relatively low pessimism (R2). Moreover, we can leverage the flexibility of HaRTES to enable the dynamic management of the real-time flows by the RTOF-enhanced SDN controller (R3). Additionally, our enhanced SDN controller exhibits the monitoring capabilities to keep track of the existing flows, current link states, and flow statistics, providing a powerful monitoring and diagnostics framework (R4). Finally, the whole network can be set up from a single entity, *i.e.* the SDN controller, using a single management protocol, *i.e.* OpenFlow, even for the RT flows with the proposed RTOF extension. The controller can also be easily extended with additional network management protocols, *e.g.* SNMP, to configure possible non-OpenFlow devices. This eases network management and allows reducing (or avoiding) the dependence on multiple vendor-specific management tools (R5).

IX. CONCLUSION

SDN is a network management paradigm that entails promising features to support Industry 4.0. However, until recently, its real-time features were not up to the stringent requirements of demanding industrial applications, particularly with the level of dynamic reconfiguration underlying Industry 4.0. Therefore, in this paper, we proposed taking the manageability, monitoring, and flexibility of SDN adding real-time support without reducing those features. We proposed extending the SDN OpenFlow protocol with real-time services, namely the Real-Time OpenFlow (RTOF) extension. We showed the RTOF architecture and the support it needs from the data plane technology. The control plane - data plane interface is carried out by a suitable OpenFlow Mediator and the OpenFlow features require an OpenFlow pipeline running in the data plane. We instantiated these components on HaRTES Ethernet switches due to their support for flexible RT communication, and we built a suitable RTOF-enabled SDN controller with open technology. The complete framework was successfully validated in practice with a prototype implementation in a realistic use-case that established the feasibility of the RTOF extension and its capacity to meet all the network requirements of Industry 4.0. In future work, we will continue enlarging the applicability of the RTOF SDN extension, applying it to more data plane technologies.

REFERENCES

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017.

[2] *Platform Industrie 4.0 Working Paper: Aspects of the Research Roadmap in Application Scenarios*, Federal Ministry Econ. Affairs Energy (BMWi), Berlin, Germany, Berlin, Germany, 2016. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf?__blob=publicationFile&v=6

[3] J.-S. Bedo. (2015). *White Paper: 5G and the Factories of the Future*. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Factories-of-the-Future-Vertical-Sector.pdf

[4] V. Koch, S. Kuge, R. Geissbauer, and S. Schrauf, *Industry 4.0: Opportunities and Challenges of the Industrial Internet*, Strategy PwC, New York, NY, USA, 2014.

[5] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[6] G. Kálmán, "Applicability of software defined networking in industrial Ethernet," in *Proc. 22nd Telecommun. Forum Telfor (TELFOR)*, Nov. 2014, pp. 340–343.

[7] D. Henneke, L. Wisniewski, and J. Jasperneite, "Analysis of realizing a future industrial network by means of software-defined networking (SDN)," in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2016, pp. 1–4.

[8] M. Ehrlich, D. Krummacker, C. Fischer, R. Guillaume, S. S. P. Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, H. D. Schotten, and J. Jasperneite, "Software-defined networking as an enabler for future industrial network management," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Automat. (ETFA)*, vol. 1, Sep. 2018, pp. 1109–1112.

[9] L. Silva, P. Gonçalves, R. Marau, P. Pedreiras, and L. Almeida, "Extending OpenFlow with flexible time-triggered real-time communication services," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[10] H. M. Hashemian, *Enterprise-Control System Integration—Part 1: Models and Terminology*, Standard ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod), 2010.

[11] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, and T. Eschert, "Industrial Internet of Things and cyber manufacturing systems," in *Industrial Internet of Things*. Cham, Switzerland: Springer, 2017, pp. 3–19, doi: 10.1007/978-3-319-42559-7_1.

[12] *Service Requirements for Next Generation New Services and Markets*, document ETSI TS 122 261 v15.5.0-5G, 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/122200_122299/122261/15.05.00_60/ts_122261v150500p.pdf

[13] W. Lepuschitz, "Self-reconfigurable manufacturing control based on ontology-driven automation agents," Ph.D. dissertation, Technische Univ. Wien, Vienna, Austria, 2018. [Online]. Available: http://repositum.tuwien.ac.at/obvutwhs/content/titleinfo/2582212?lang=en

[14] T. Bangemann, M. Riedl, M. Thron, and C. Diedrich, "Integration of classical components into industrial cyber–physical systems," *Proc. IEEE*, vol. 104, no. 5, pp. 947–959, May 2016.

[15] D. Thiele and R. Ernst, "Formal analysis based evaluation of software defined networking for time-sensitive Ethernet," in *Proc. Design, Autom. Test Eur. Conf. (DATE)*, Mar. 2016, pp. 31–36.

[16] M. Herlich, J. L. Du, F. Schörghofer, and P. Dorfinger, "Proof-of-concept for a software-defined real-time Ethernet," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–4.

[17] C. Ternon, J. Goossens, and J.-M. Dricot, "FTT-openFlow, on the way towards real-time SDN," *ACM SIGBED Rev.*, vol. 13, no. 4, pp. 49–54, Nov. 2016.

[18] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. ACM 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, New York, NY, USA, 2016, pp. 193–202.

[19] K. Ahmed, J. O. Blech, M. A. Gregory, and H. Schmidt, "Software defined networking for communication and control of cyber-physical systems," in *Proc. IEEE 21st Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2015, pp. 803–808.
[20] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Proc. 2nd Eur. Workshop Softw. Defined Netw.*, Oct. 2013, pp. 81–86.
[21] Open Networking Foundation. (Dec. 2014). *OpenFlow Switch Specification Version 1.5.0 (Protocol Version 0x06)*. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[22] R. Santos, "Enhanced Ethernet switching technology for adaptive hard real-time applications," Ph.D. dissertation, Dept. Eletrón., Univ. Aveiro, Aveiro, Portugal, 2011. [Online]. Available: http://hdl.handle.net/10773/7142
[23] NetFPGA ORG. *NetFPGA Home Page*. Accessed: Jan. 6, 2019. [Online]. Available: http://netfpga.org/site/#/
[24] M. Ashjaei, L. Silva, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte, "Improved message forwarding for multi-hop HaRTES real-time Ethernet networks," *J. Signal Process. Syst.*, vol. 84, no. 1, pp. 47–67, Jul. 2016.
[25] CPqD. *OpenFlow Software Switch 1.3*. Accessed: Oct. 13, 2018. [Online]. Available: http://cpqd.github.io/ofsoftswitch13/
[26] P. A. Ribeiro, L. Duoba, R. Prior, S. Crisostomo, and L. Almeida, "Real-time wireless data plane for real-time-enabled SDN," in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2019, pp. 1–4.
[27] L. Silva, P. Pedreiras, P. Fonseca, and L. Almeida, "On the adequacy of SDN and TSN for Industry 4.0," in *Proc. IEEE 22nd Int. Symp. Real-Time Comput. (ISORC)*, May 2019, pp. 43–51.
[28] BSN. *Floodlight*. Accessed: Oct. 13, 2018. [Online]. Available: https://www.projectfloodlight.org
[29] REANNZ. *Faucet*. Accessed: Oct. 13, 2018. [Online]. Available: https://faucet.nz/
[30] The Linux Foundation. *OpenDaylight*. Accessed: Apr. 16, 2019. [Online]. Available: https://www.opendaylight.org/
[31] Open Network Foundation. *Open Network Operating System (ONOS)*. Accessed: Apr. 16, 2019. [Online]. Available: https://onosproject.org/
[32] Ryu SDN Framework Community. *RYU SDN Framework*. Accessed: Apr. 16, 2019. [Online]. Available: https://osrg.github.io/ryu/
[33] NetworkX Community. *Networkx: Software for Complex Networks*. Accessed: Oct. 10, 2018. [Online]. Available: https://networkx.github.io/
[34] C. Liu, F. Li, G. Chen, and X. Huang, "TTEthernet transmission in software-defined distributed robot intelligent control system," *Wireless Commun. Mobile Comput.*, vol. 2018, Jul. 2018, Art. no. 8589343.

LUIS MOUTINHO was born in Fermelã, Aveiro, Portugal, in 1987. He received the M.Sc. degree in electronics and telecommunications engineering and the Ph.D. degree in telecommunications from the University of Aveiro, Portugal, in 2010 and 2019, respectively. He is currently an invited Adjunct Professor with the Escola Superior de Tecnologia e Gestão de Águeda (ESTGA), Portugal, and a Research Assistant with the Instituto de Telecomunicações (IT), Portugal. He published a book chapter as well as over 15 articles in conferences and journals related to his domains of interest, particularly real-time communications for industrial systems, software-defined networking, and vehicular networks. He was a member of the organizing committee of the 12th IEEE World Conference on Factory Communication Systems (WFCS), Portugal, in 2016, and a Workshop Chair at the 1st EAI International Conference on Future Intelligent Vehicular Technologies (Future5V), Portugal, in 2016.

PAULO PEDREIRAS graduated in electronics and telecommunications engineering, in 1997, and received the Ph.D. degree in electrotechnical engineering from the University of Aveiro, Portugal, in 2003. He is currently an Assistant Professor with the Electronics, Telecommunications and Informatics Department, University of Aveiro. He is with the Portuguese Telecommunications Institute, Aveiro, coordinating the Embedded Systems Group-AV. His current research interests include real-time embedded systems, wireless communications, electrical instrumentation, and industrial communications. He participated in more than 15 national and European research projects, with coordination responsibilities at diverse levels. Since 2000, he has authored or coauthored over 150 articles in international peer-reviewed conferences and journals. He participates regularly on the technical program committees of some of the more relevant events of his research area, such as WFCS, SIES, and ETFA. He collaborates regularly, as Reviewer, in several top international journals, such as the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, *Journal of Systems Architecture* (Elsevier), and Springer *Real-Time Systems* Journal. He participated on the organization of several international scientific events, such as a Program Co-Chair in WFCS2015, a General Co-Chair in WFCS2016, a Track Chair in ETFA2017-T2, WFCS2017 and WFCS2018 (Steering Committee), and a Workshop Co-Chair in ETFA2019.

LUIS ALMEIDA graduated in electronics and telecommunications engineering, in 1988, and received the Ph.D. degree in electrical engineering, in 1999, from the University of Aveiro, Portugal. He is currently an Associate Professor with the Electrical and Computer Engineering Department, University of Porto (UP), Portugal, where he coordinates the Distributed and Real-time Embedded Systems Laboratory (DaRTES). He is also a Vice-Director of the CISTER Research Center on Real-Time and Embedded Computing Systems and a Vice-Chair of the IEEE Technical Committee on Real-Time Systems. He published over 300 articles in related conferences and journals. He was a Program and General Chair of the IEEE Real-Time Systems Symposium, in 2011 and 2012, respectively, a Trustee of the RoboCup Federation, from 2008 to 2016, and a Vice-President, from 2011 to 2013. He is an Associate Editor of the Springer Journal of *Real-Time Systems*, the Elsevier *Journal of Systems Architecture*, and the SAGE *International Journal on Advanced Robotic Systems*. He regularly participates in the organization of scientific events in his domains of interest, namely real-time communications for distributed industrial/embedded systems, for teams of cooperating agents and for sensor networks.